在笔者的上一篇文章 《驱动开发：内核特征码扫描PE代码段》 中 LyShark 带大家通过封装好的

LySharkToolsUtilKernelBase 函数实现了动态获取内核模块基址，并通过 ntimage.h 头文件中提供 的系列函数解析了指定内核模块的 PE节表 参数，本章将继续延申这个话题，实现对PE文件导出表的解析任务，导出表无法动态获取，解析导出表则必须读入内核模块到内存才可继续解析，所以我们需要分两 步走，首先读入内核磁盘文件到内存，然后再通过 ntimage.h 中的系列函数解析即可。

当PE文件执行时Windows装载器将文件装入内存并将导入表中登记的DLL文件一并装入，再根据DLL文 件中函数的导出信息对可执行文件的导入表(IAT)进行修正。导出函数在DLL文件中，导出信息被保存在导出表，导出表就是记载着动态链接库的一些导出信息。通过导出表，DLL文件可以向系统提供导出函数的 名称、序号和入口地址等信息，以便Windows装载器能够通过这些信息来完成动态链接的整个过程。

导出函数存储在PE文件的导出表里，导出表的位置存放在PE文件头中的数据目录表中，与导出表对应的 项目是数据目录中的首个 IMAGE\_DATA\_DIRECTORY 结构，从这个结构的 VirtualAddress 字段得到的就是导出表的RVA值，导出表同样可以使用函数名或序号这两种方法导出函数。

导出表的起始位置有一个 IMAGE\_EXPORT\_DIRECTORY 结构，与导入表中有多个

IMAGE\_IMPORT\_DESCRIPTOR 结构不同，导出表只有一个 IMAGE\_EXPORT\_DIRECTORY 结构，该结构定义如下：



} IMAGE\_EXPORT\_DIRECTORY, PIMAGE\_EXPORT\_DIRECTORY;

// 指向文件名的RVA

// 导出函数的起始序号

// 导出函数总数

// 以名称导出函数的总数

// 导出函数地址表的RVA

// 函数名称地址表的RVA AddressOfNameOrdinals; // 函数名序号表的RVA

// 文件的产生时刻

Characteristics; TimeDateStamp; MajorVersion; MinorVersion; Name;

Base; NumberOfFunctions; NumberOfNames; AddressOfFunctions;

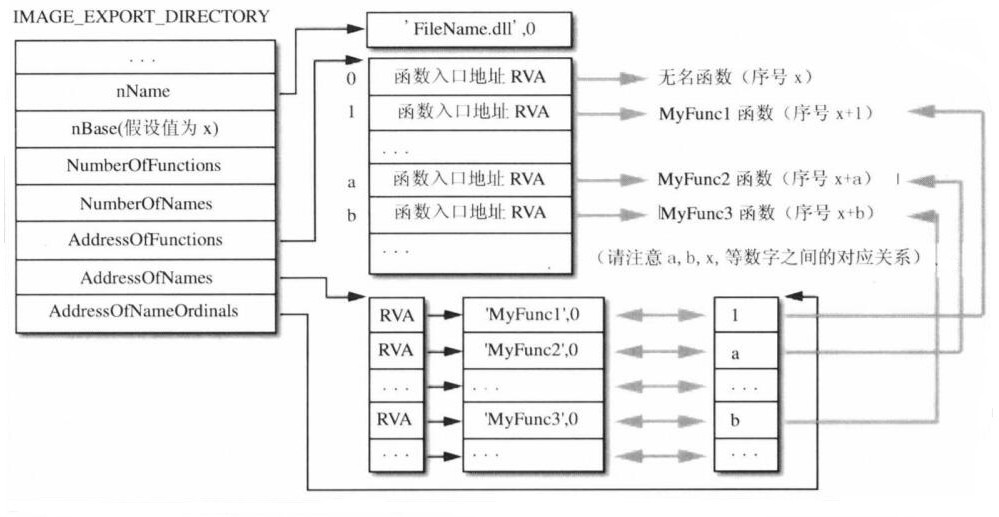
AddressOfNames;

DWORD DWORD WORD WORD DWORD DWORD DWORD DWORD DWORD DWORD

DWORD

typedef struct \_IMAGE\_EXPORT\_DIRECTORY {

上面的 \_IMAGE\_EXPORT\_DIRECTORY 结构如果总结成一张图，如下所示：



在上图中最左侧 AddressOfNames 结构成员指向了一个数组，数组里保存着一组RVA，每个RVA指向一个字符串即导出的函数名，与这个函数名对应的是 AddressOfNameOrdinals 中的结构成员，该对应项存储的正是函数的唯一编号并与 AddressOfFunctions 结构成员相关联，形成了一个导出链式结构体。

获取导出函数地址时，先在 AddressOfNames 中找到对应的名字 MyFunc1 ，该函数在 AddressOfNames 中是第1项，然后从 AddressOfNameOrdinals 中取出第1项的值这里是1，然后就可以通过导出函数的序号 AddressOfFunctions[1] 取出函数的入口RVA，然后通过RVA加上模块基址便是第一个导出函数的地址，向后每次相加导出函数偏移即可依次遍历出所有的导出函数地址。

其解析过程与应用层基本保持一致，如果不懂应用层如何解析也可以去看我以前写过的 《PE格式：手写PE

结构解析工具》 里面具体详细的分析了解析流程。

首先使用 InitializeObjectAttributes() 打开文件，打开后可获取到该文件的句柄，

InitializeObjectAttributes 宏初始化一个 OBJECT\_ATTRIBUTES 结构体, 当一个例程打开对象时由此结构体指定目标对象的属性，此函数的微软定义如下；

);

// 权限

// 文件名

// 输出文件

// 权限

// 0

VOID InitializeObjectAttributes(

[out] POBJECT\_ATTRIBUTES p,

[in] PUNICODE\_STRING n,

[in] ULONG a,

[in] HANDLE r,

[in, optional] PSECURITY\_DESCRIPTOR s

当权限句柄被初始化后则即调用 ZwOpenFile() 打开一个文件使用权限 FILE\_SHARE\_READ 打开，打开文件函数微软定义如下；

FILE\_SYNCHRONOUS\_IO\_NONALERT。

);

// 打开选项，一般设为

OpenOptions

// 共享的权限。可以是FILE\_SHARE\_READ

ShareAccess,

// 返回打开文件的句柄

// 打开的权限，一般设为GENERIC\_ALL。

// OBJECT\_ATTRIBUTES结构

// 指向一个结构体的指针。该结构体指明打

IoStatusBlock,

[out] PIO\_STATUS\_BLOCK

开文件的状态。

[in] ULONG

或者 FILE\_SHARE\_WRITE。[in] ULONG

NTSYSAPI NTSTATUS ZwOpenFile(

[out] PHANDLE FileHandle,

[in] ACCESS\_MASK DesiredAccess, [in] POBJECT\_ATTRIBUTES ObjectAttributes,

接着文件被打开后，我们还需要调用 ZwCreateSection() 该函数的作用是创建一个 Section 节对象， 并以PE结构中的 SectionALignment 大小对齐映射文件，其微软定义如下；

|  |  |  |  |
| --- | --- | --- | --- |
| [in] ULONG | AllocationAttributes, | // | 指定确定节的分配属 |
| 性的SEC\_XXX 标志的位掩码。  [in, optional] HANDLE | FileHandle | // | （可选）指定打开的 |

最后读取导出表就要将一个磁盘中的文件映射到内存中，内存映射核心文件时 ZwMapViewOfSection() 该系列函数在应用层名叫 MapViewOfSection() 只是一个是内核层一个应用层，这两个函数参数传递基本一致，以 ZwMapViewOfSection 为例，其微软定义如下；

NTSYSAPI NTSTATUS ZwCreateSection(

[out] PHANDLE SectionHandle,

// 指向 HANDLE 变量的

指针，该变量接收 section 对象的句柄。

[in] ACCESS\_MASK DesiredAccess, // 指定一个

ACCESS\_MASK 值，该值确定对 对象的请求访问权限。

[in, optional] POBJECT\_ATTRIBUTES ObjectAttributes, // 指向

OBJECT\_ATTRIBUTES 结构的指针，该结构指定对象名称和其他属性。

[in, optional] PLARGE\_INTEGER

（以字节为单位）。

MaximumSize,

// 指定节的最大大小

[in]

页面上放置的保护。

ULONG

SectionPageProtection,

// 指定要在 节中的每个

文件对象的句柄。

);



// 权限

Win32Protect

ULONG

// 指定填充地址

// 0

// 每次提交大小 1024

// 0

// 浏览大小

// ViewShare

// 分配类型

AllocationType,

ULONG

SECTION\_INHERIT InheritDisposition,

BaseAddress, ZeroBits, CommitSize, SectionOffset,

ViewSize,

PSIZE\_T

[in, out] [in]

[in]

MEM\_TOP\_DOWN

[in]

PAGE\_READWRITE(读写)

);

NtCurrentProcess()获取自身句柄

[in, out] PVOID

[in] ULONG\_PTR

[in] SIZE\_T

[in, out, optional] PLARGE\_INTEGER

// 接收一个节对象

// 进程句柄,此处使用

NTSYSAPI NTSTATUS ZwMapViewOfSection(

[in] HANDLE SectionHandle,

[in] HANDLE ProcessHandle,

将如上函数研究明白那么代码就变得很容易了，首先 InitializeObjectAttributes 设置文件权限与属性，然后调用 ZwOpenFile 打开文件，接着调用 ZwCreateSection 创建节对象，最后调用

ZwMapViewOfSection 将磁盘文件映射到内存，这段代码实现起来很简单，完整案例如下所示；



// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

#include <ntifs.h> #include <ntimage.h> #include <ntstrsafe.h>

// 内存映射文件

NTSTATUS KernelMapFile(UNICODE\_STRING FileName, HANDLE phFile, HANDLE phSection, PVOID ppBaseAddress)

{



NTSTATUS status = STATUS\_SUCCESS; HANDLE hFile = NULL;

HANDLE hSection = NULL; OBJECT\_ATTRIBUTES objectAttr = { 0 }; IO\_STATUS\_BLOCK iosb = { 0 };

PVOID pBaseAddress = NULL;

SIZE\_T viewSize = 0;

// 设置文件权限

InitializeObjectAttributes(&objectAttr, &FileName, OBJ\_CASE\_INSENSITIVE | OBJ\_KERNEL\_HANDLE, NULL, NULL);

// 打开文件

status = ZwOpenFile(&hFile, GENERIC\_READ, &objectAttr, &iosb, FILE\_SHARE\_READ, FILE\_SYNCHRONOUS\_IO\_NONALERT);

if (!NT\_SUCCESS(status))

{

return status;

}

// 创建节对象

status = ZwCreateSection(&hSection, SECTION\_MAP\_READ | SECTION\_MAP\_WRITE, NULL, 0, PAGE\_READWRITE, 0x1000000, hFile);

if (!NT\_SUCCESS(status))

{

ZwClose(hFile); return status;

}

// 映射到内存

status = ZwMapViewOfSection(hSection, NtCurrentProcess(), &pBaseAddress, 0, 1024, 0, &viewSize, ViewShare, MEM\_TOP\_DOWN, PAGE\_READWRITE);

if (!NT\_SUCCESS(status))

{

ZwClose(hSection); ZwClose(hFile); return status;

}

// 返回数据

phFile = hFile; phSection = hSection;

ppBaseAddress = pBaseAddress;

return status;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载 \n");

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark.com \n");

NTSTATUS status = STATUS\_SUCCESS;

HANDLE hFile = NULL;

HANDLE hSection = NULL; PVOID pBaseAddress = NULL;

UNICODE\_STRING FileName = {0};

// 初始化字符串

RtlInitUnicodeString(&FileName, L"\\??

\\C:\\Windows\\System32\\ntoskrnl.exe");

// 内存映射文件

status = KernelMapFile(FileName, &hFile, &hSection, &pBaseAddress); if (NT\_SUCCESS(status))

{

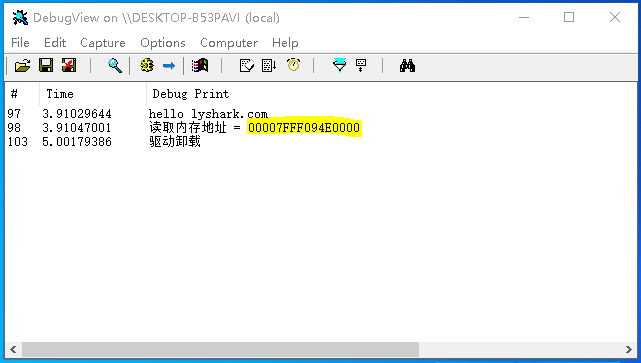
DbgPrint("读取内存地址 = %p \n", pBaseAddress);

}

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

运行这段程序，即可读取到 ntoskrnl.exe 磁盘所在文件的内存映像基地址，效果如下所示；



如上代码读入了 ntoskrnl.exe 文件，接下来就是解析导出表，首先将 pBaseAddress 解析为

PIMAGE\_DOS\_HEADER 获取DOS头，并在DOS头中寻找 PIMAGE\_NT\_HEADERS 头，接着在 NTHeader 头中得到数据目录表，此处指向的就是导出表 PIMAGE\_EXPORT\_DIRECTORY 通过 pExportTable-

>NumberOfNames 可得到导出表的数量，通过 (PUCHAR)pDosHeader + pExportTable-

>AddressOfNames 得到导出表的地址，依次循环读取即可得到完整的导出表。

// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{



DbgPrint("hello lyshark.com \n");

NTSTATUS status = STATUS\_SUCCESS; HANDLE hFile = NULL;

HANDLE hSection = NULL; PVOID pBaseAddress = NULL;

UNICODE\_STRING FileName = { 0 };

LONG FunctionIndex = 0;

// 初始化字符串

RtlInitUnicodeString(&FileName, L"\\??

\\C:\\Windows\\System32\\ntoskrnl.exe");

// 内存映射文件

status = KernelMapFile(FileName, &hFile, &hSection, &pBaseAddress); if (NT\_SUCCESS(status))

{

DbgPrint("[LyShark] 读取内存地址 = %p \n", pBaseAddress);

}

// Dos 头

PIMAGE\_DOS\_HEADER pDosHeader = (PIMAGE\_DOS\_HEADER)pBaseAddress;

// NT 头

PIMAGE\_NT\_HEADERS pNtHeaders = (PIMAGE\_NT\_HEADERS)((PUCHAR)pDosHeader +

pDosHeader->e\_lfanew);

// 导出表

PIMAGE\_EXPORT\_DIRECTORY pExportTable = (PIMAGE\_EXPORT\_DIRECTORY)

((PUCHAR)pDosHeader + pNtHeaders-

>OptionalHeader.DataDirectory[0].VirtualAddress);

// 有名称的导出函数个数

ULONG ulNumberOfNames = pExportTable->NumberOfNames;

DbgPrint("[LyShark.com] 导出函数个数: %d \n\n", ulNumberOfNames);

// 导出函数名称地址表

PULONG lpNameArray = (PULONG)((PUCHAR)pDosHeader + pExportTable-

>AddressOfNames);

PCHAR lpName = NULL;

// 开始遍历导出表(输出ulNumberOfNames导出函数) for (ULONG i = 0; i < ulNumberOfNames; i++)

{

lpName = (PCHAR)((PUCHAR)pDosHeader + lpNameArray[i]);

// 获取导出函数地址

USHORT uHint = (USHORT )((PUCHAR)pDosHeader + pExportTable-

>AddressOfNameOrdinals + 2 i);

ULONG ulFuncAddr = (PULONG)((PUCHAR)pDosHeader + pExportTable-

>AddressOfFunctions + 4 uHint);

PVOID lpFuncAddr = (PVOID)((PUCHAR)pDosHeader + ulFuncAddr);

// 获取SSDT函数Index

FunctionIndex = (ULONG )((PUCHAR)lpFuncAddr + 4);

DbgPrint("序号: [ %d ] | Hint: %d | 地址: %p | 函数名: %s \n", i, uHint, lpFuncAddr, lpName);

}

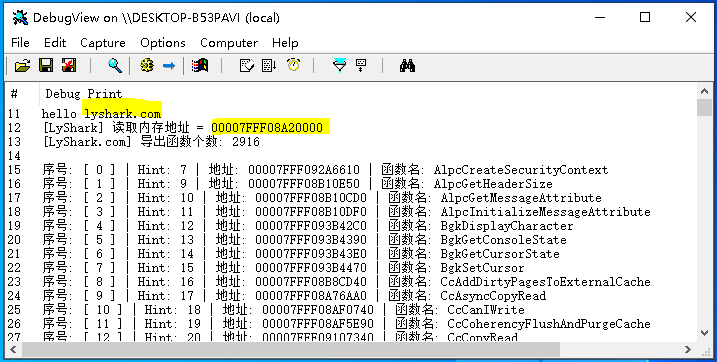
// 释放指针

ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress); ZwClose(hSection);

ZwClose(hFile);

Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

代码运行后即可获取到当前 ntoskrnl.exe 程序中的所有导出函数，输出效果如下所示； SSDT 表 通 常 会 解 析 \\??\\C:\\Windows\\System32\\ntoskrnl.exe SSSDT表通常会解析 \\??\\C:\\Windows\\System32\\win32k.sys

根据上方的函数流程将其封装为 GetAddressFromFunction() 用户传入 DllFileName 指定的PE文件， 以及需要读取的 pszFunctionName 函数名，即可输出该函数的导出地址。

// 署名权

// right to sign one's name on a piece of work

// PowerBy: LyShark

// Email: [me@lyshark.com](mailto:me@lyshark.com)

// 寻找指定函数得到内存地址

ULONG64 GetAddressFromFunction(UNICODE\_STRING DllFileName, PCHAR pszFunctionName)

{

NTSTATUS status = STATUS\_SUCCESS; HANDLE hFile = NULL;

HANDLE hSection = NULL; PVOID pBaseAddress = NULL;

// 内存映射文件

status = KernelMapFile(DllFileName, &hFile, &hSection, &pBaseAddress); if (!NT\_SUCCESS(status))

{



return 0;

}

PIMAGE\_DOS\_HEADER pDosHeader = (PIMAGE\_DOS\_HEADER)pBaseAddress; PIMAGE\_NT\_HEADERS pNtHeaders = (PIMAGE\_NT\_HEADERS)((PUCHAR)pDosHeader +

pDosHeader->e\_lfanew);

PIMAGE\_EXPORT\_DIRECTORY pExportTable = (PIMAGE\_EXPORT\_DIRECTORY)

((PUCHAR)pDosHeader + pNtHeaders-

>OptionalHeader.DataDirectory[0].VirtualAddress);

ULONG ulNumberOfNames = pExportTable->NumberOfNames;

PULONG lpNameArray = (PULONG)((PUCHAR)pDosHeader + pExportTable-

>AddressOfNames);

PCHAR lpName = NULL;

for (ULONG i = 0; i < ulNumberOfNames; i++)

{

lpName = (PCHAR)((PUCHAR)pDosHeader + lpNameArray[i]);

USHORT uHint = (USHORT )((PUCHAR)pDosHeader + pExportTable-

>AddressOfNameOrdinals + 2 i);

ULONG ulFuncAddr = (PULONG)((PUCHAR)pDosHeader + pExportTable-

>AddressOfFunctions + 4 uHint);

PVOID lpFuncAddr = (PVOID)((PUCHAR)pDosHeader + ulFuncAddr);

if (\_strnicmp(pszFunctionName, lpName, strlen(pszFunctionName)) == 0)

{

ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress); ZwClose(hSection);

ZwClose(hFile);

return (ULONG64)lpFuncAddr;

}

}

ZwUnmapViewOfSection(NtCurrentProcess(), pBaseAddress); ZwClose(hSection);

ZwClose(hFile);

return 0;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint("驱动卸载 \n");

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark.com \n");

UNICODE\_STRING FileName = { 0 };

ULONG64 FunctionAddress = 0;

// 初始化字符串

RtlInitUnicodeString(&FileName, L"\\??\\C:\\Windows\\System32\\ntdll.dll");

// 取函数内存地址

FunctionAddress = GetAddressFromFunction(FileName, "ZwQueryVirtualMemory"); DbgPrint("ZwQueryVirtualMemory内存地址 = %p \n", FunctionAddress);

Driver->DriverUnload = UnDriver;

return STATUS\_SUCCESS;

}

如上程序所示，当运行后即可获取到 ntdll.dll 模块内 ZwQueryVirtualMemory 的导出地址，输出效果如下所示；

